Object-Oriented concepts 6

Today we are in the era of Internet, Websites and Web based operations, where rapid application development and reusability of source code is very important. Object-oriented techniques as methodology or as paradigm is playing significant role in analysis, design and implementation of software system. Software developed using object-oriented methodology are proclaimed to be more reliable, easier to maintain, reuse and enhance. This chapter explains basic object-oriented concepts. Implementation of this concepts using Java programming language is covered in the later chapters.

Introduction

Object-oriented programming concepts started originating in the 1960s. Since mid 1980s, it had become the main programming paradigm used in the creation of new software. It was developed as a way to handle the rapidly increasing size and complexity of software systems and to make it easier to modify these large and complex systems over time. Some of the popular programming languages that support object-oriented programming are C++, Java, C#, VB.net, ASP.net and PHP.

The way of programming can be divided into two categories namely structure/procedural and object-oriented. In procedural programming; the focus is on writing functions or procedures which operate on data. For example, for library application software, we will think of all processes of library application and focus on the modules like student registration, book issue, book return, fine calculation.

In object-oriented paradigm, the focus is on objects which contain both data and functionality together. For library application, our focus is on the objects involved in the application. Here we think of objects like student, book and librarian. We also need to think of association between such objects. For example, student returns book.

The power of object-oriented programming language enables the programmer to create modular, reusable and extendable code. As a result, programmer can formulate a program by composition and modification of existing modules. Flexibility is gained by being able to change or replace modules without disturbing other parts of code. Software development speed is gained by reusing and enhancing the existing code.

Object-oriented programming uses object as its fundamental building block. Similar objects are classified using a concept of class. A computer language is object-oriented if they support four specific object properties called abstraction, encapsulation, polymorphism and inheritance.

Object

In the "real" world, objects are the entities of which the world is comprised. Some objects can be concrete things like person, car or a coffee cup. Other objects may be abstract that do not represent things which can be touched or seen; for example, concepts like date and time.

All objects have unique identity and are distinguishable from each other. For example, every person has characteristics like name, city, gender, birth-date, profession. In object-oriented terminology, such

characteristics are known as **properties** or **attributes**. To uniquely distinguish one person from other, we use the value of these characteristics. The names 'Ram' and 'Shyam' specify two different persons. When two persons have same name, they may be distinguished using other attribute like birth-date. In this way, to identify the objects, we use the value of these attributes. These values are called **state**. Additionally, there is a **behaviour** associated with objects. For example, person takes birth, gets name, changes location. The behaviour is also known as **method**. State of the object can change due to its behaviour. Thus, any real world object can be described in terms of what it is called (identity), what it is (its state) and what it does (its behaviour).

In object-oriented programming, attributes that describe the object are also referred to as data fields. The data attributes and behavioral methods associated with an object are collectively referred to as its **members *or* features.**

When we design for any software application, objects to be considered should be meaningful to the application. For example, in railway reservation application, meaningful objects are train, passenger, ticket or station. Objects like car, computer and watch may be irrelevant here.

Class

From the example of the object person, it can be easily seen that various objects possessing same characteristics and behavior differ from each other only in the state (value of the data) that they hold. Object-oriented system uses the concept of class that enables to express the set of objects that are abstractly equivalent, differing only in the values of their attributes. Class can be considered as a blueprint for various objects.

A class is a template for multiple objects with similar features. It describes a group of objects with similar attributes and common behavior. Objects in the same class share a common semantic purpose. Thus, class is a general concept used to embody all the common features of a particular set of objects.

For example, we have a class named 'Person' describing common attributes and behavior of all persons. Individual persons are then objects and are identified by the state; that is the value of their attributes. Figure 6.1 shows difference between the class and its objects.

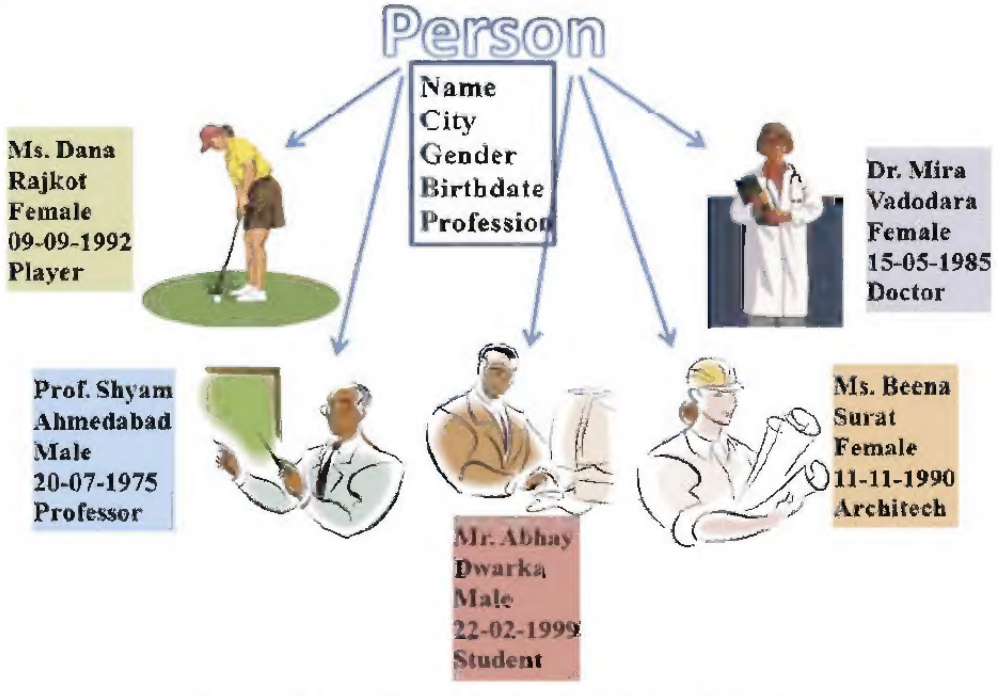

Figure 6.1 : Class 'Person' and its Objects

Before we proceed ahead to learn other object-oriented concepts, let us have brief introduction to class diagram.

Introduction to Class Diagram

The class diagram presents a collection of classes, constraints and relationship among classes.

Unified Modelling Language (UML) can be used to create models of object-oriented software to help with design of an application. UML is a visual modelling language defined and maintained by the Object Management Group (OMG). UML specifies several diagrams for representing different aspects of a software application.

The purpose of the class diagram is to model the static view of an application. The class diagrams are the only diagrams which can be directly mapped with object oriented languages and thus widely used among software developers.

In class diagram, a class is represented with an icon using a rectangle split into three sections to present name, attributes and behavior as follows :

1. Name of the class in the top section
2. Attributes or properties of the class in the middle section
3. Behavior or operations or methods of the class in the bottom section

Figure 6.2 shows the pictorial representation of a class using UML convention, while figure 6.3 shows the class diagram of Person.

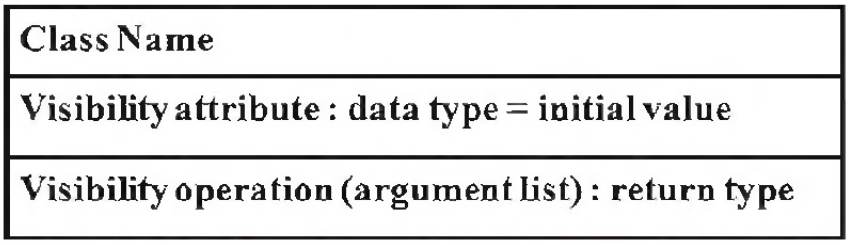

Figure 6.2 : Class representation in UML convention

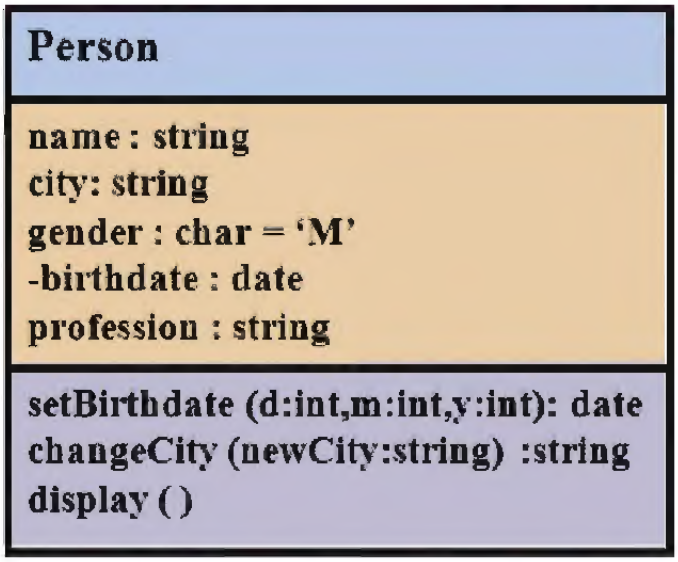

Figure 6.3 : Diagram of class 'Person'

The class Person is intended to provide information that is useful in understanding the role of the class in the context of class diagram. It does not have to contain every attribute and operation of the class.

As seen in figure 6.2, in UML notation, an attribute is declared using following syntax :

[<visibility>] <attribute name> [: <attribute data type> [= <initial value>]]

Here, items written in pair of square brackets [ ] are optional and user is supposed to specify the values for items enclosed in angle brackets < >. Visibility can be private, protected, public or package represented using symbols -, #, + and ~ respectively. We will study about visibility in later chapters. Attribute generally refers to a variable. Data type and initial value identify the type of data stored and its value at the start of the program. As can be seen here, only attribute-name is mandatory and all other items are optional.

Some examples of declaring attribute are as follows :

name : string

- balance : float = 0.0

As seen in figure 6.2, In UML notation, an operation is declared using following syntax :

[<visibility>] <method name> (parameter list separated by comma) : <return data type>

An example of a method in the previous example is setBirthdate(d:int, m:int, y:int) : date. Here the parameter can be specified with data type and optional default value; for example gender : char = 'M'. Parameters can also be specified as input or output depending on whether it is read only or not.

UML diagrams are independent of the programming language used for coding an application. Some software developers prefer to specify attributes and operations in the more familiar format of a programming language instead of using the standard UML notation. That's fine, as long as it makes sense to everyone concerned.

Objects are presented using their state during execution of an application. Thus, objects are dynamic. Corresponding to figure 6.1, an object (also called an instance) of class person is represented as shown in figure 6.4.

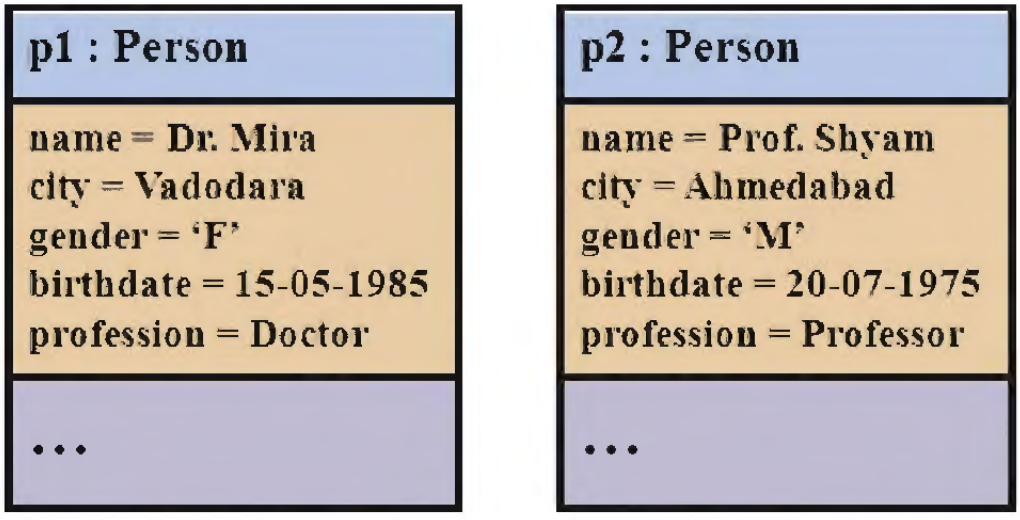

Figure 6.4 : Objects p1 and p2 of Class 'Person'

Encapsulation

For any computer program, two core elements are data and functions. Structured/Procedural programming views these two core elements as two separate entities; whereas object-oriented programming views them as single entity.

In procedural programming, data can be altered by any component of the program. It is not protected from modification.

In object-oriented programming, this problem can be solved using encapsulation. Here, data and the methods that manipulate data are guarded against modification or misuse by other components of the program. This mechanism of providing protection to data and methods of a program is called **encapsulation.** This is possible by wrapping data and methods into a single unit known as class and declaring them as private. Private members of the class are not available directly to outside world. If necessary, the data is made available via public methods. Thus, encapsulation provides data hiding capability. We will study about class and private/public data and methods in later chapters.

Encapsulation keeps the data safe from unintended actions and inadvertent access by outside objects. Unlike procedural programming where common data areas are often used for sharing information, object-oriented programming discourages direct access to common data (other than the use of global variables) by other programs. Only the object that "owns" the data can change its content. Other objects can view or change this data by sending message to the "owner".

Data Abstraction

Data abstraction is a process of representing the essential features of the objects without including implementation detail. Abstraction is a concept that hides the complexity; it says what it does, but not how it is done.

Let's take one real life example of a television set. We can turn television set on and off, change the channel, adjust the volume. But, we do not know its internal details like how it receives signals over the air or through a cable, how it translates them and finally displays them on the screen. Thus there is a clear separation of internal implementation of television set from its external interface. We can play with it via external interfaces like the power button, channel changer and volume control without having any knowledge of its internals.

Data abstraction thus is a technique that relies on the separation of interface and implementation. It is not a new concept in programming. Most people already practice it up-to some degree, probably without realizing it. For example, when we use functions like sqrt(25) and printf("Hello world"), in C programming we never thought of how they are implemented.

A user-defined function with necessary input data parameters also provides data abstraction. Let us consider a C program using structure for declaring 'date' type of data. Here date is composed of three elements; day, month and year. Let us use primitive integer data type for these elements. Consider a function named 'dateDiff' that takes two dates as parameters and returns the number of days between two dates. User of this function is not supposed to know how this difference is calculated. It is necessary to know only the signature of the function; that is the name of the function, number and type of parameters and the return type of parameters. If there is any change in the process of finding difference, it has no effect on the part of the program using this function.

Thus data abstraction provides the skeleton or templates for our use. The system hides certain details of how data is stored, created and maintained. All that is visible to the rest of the world is the

abstract behaviour of the data type; details of how that behaviour is implemented are hidden, so that they can be changed as per need without impacting others.

Abstract Data Types (ADT) or structures (struct) in C/C++, classes in C++/Java are examples for data abstraction. In ADT, we simply define a data type and a set of operations on it. We do not show the implementation of operations.

The basic difference between data encapsulation and data abstraction is: Encapsulation protects data by making them inaccessible from outside and abstraction enables to represent data in which the implementation details are hidden (abstracted).

Messaging

In object-oriented terminology, a call to a method is referred to as a message. Due to encapsulation, all method calls are handled by objects that recognize the method.

Different classes may have same methods with same name. For example, class 'date' has method named 'display' that displays the date object in specific format. Suppose there are other classes 'time' and 'person'; both having method with same name 'display'; to display time in specific format and to display the details of person respectively. When method 'display' is invoked in the program, how to know which method is to be executed? This is determined using the object that invokes the method. For example, if 'display' is called by an object of 'person' class, it executes the display method defined in 'person' class.

Polymorphism

Polymorphism means 'many forms'. There may be different forms of single method or operation.

Assume that we have written a function named 'max' that finds maximum of two numbers. It takes two integers as parameters and returns maximum as integer value. Now suppose we want to add one more function named 'max' that finds maximum of integer elements stored in an array. It takes array and its size as parameters and return maximum integer. Is it possible to define more than one function with the same name ? In some programming languages, the answer to this question is 'no'. But in object-oriented programming, the answer is 'yes' as long as the methods differs in signatures (number and type of parameters).

Object-oriented programming allows defining more than one method having same name but different signatures in a single class. This feature is known as **function** or **method overloading.**

Object-oriented programming also allows writing expression using operators on objects. For example, we can use expression 'date1-date2' where both operands are objects of class 'date'. Here operation '-' is performed in a way different than performing subtraction on two numbers; say n1-n2. Here same operation is given different meanings depending upon the data type of operands used. This type of polymorphism is achieved through **operator overloading.**

Thus, polymorphism is achieved using two types of overloading: function overloading and operator overloading. In general, the capability of using same names to mean different things in different contexts is called overloading.

Aggregation and Composition

When objects of one class are composed of objects of other class, it is called aggregation or composition. It represents 'has-a' or 'a-part-of relationship between classes. For example, motherboard is a part of computer.

As we know, computer is composed of many parts like motherboard, screen, keyboard and mouse. A screen itself may be a class with attributes like length, width and model. Similarly motherboard may be defined as a class with attributes like model and company. When we define class 'computer', it will contain attributes that are objects of class 'motherboard' and 'screen'. Thus it implies containment.

Let us modify class 'Person' discussed before. First of all, we will define two new classes 'Name' and 'Address' as shown in figure 6.5. Class 'Name' has attributes first name, middle name and last name. Class 'Address' has attributes house (that refers to details of house), street, city, state and pin code. Now let us modify class 'Person' and change the name of the attributes name and address to nm and addr respectively. The data type of attributes nm and addr is class 'Name' and 'Address' respectively. Thus class 'Person' contains objects of class 'Name' and 'Address'.

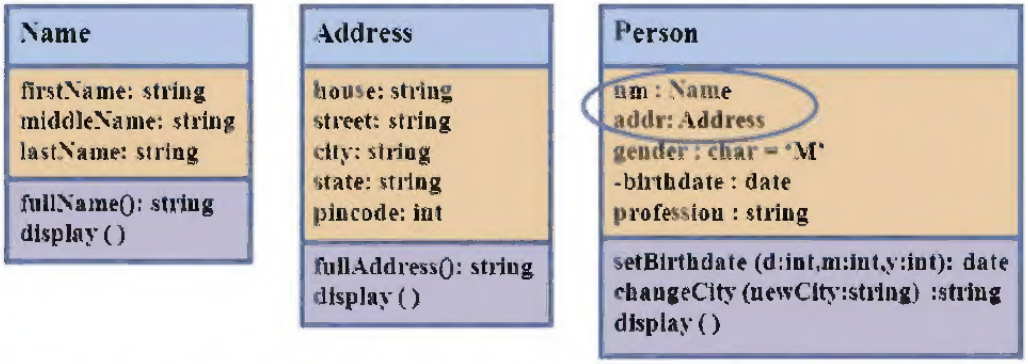

Figure 6.5 : Class 'Person' with attributes of class 'Name' and 'Address'

Aggregation vs. Composition

Aggregation represents non-exclusive relationship between two classes. In aggregation, the class that forms part of the owner class can exist independently. The life of an object of the part class is not determined by the owner class. For example, although the motherboard is part of the computer, it can exist as a separate item independent of the computer. Thus motherboard is not exclusively associated with computer. Similarly object address may be shared by two or more persons as can be seen in figure 6.6. So, address is not exclusive to any one person.

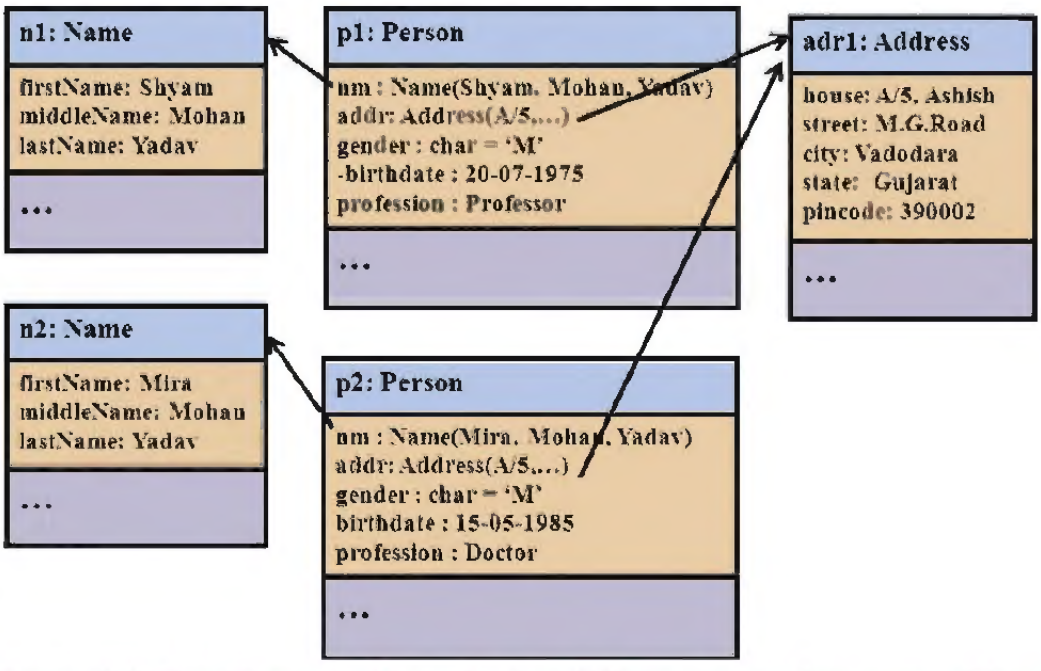

Figure 6.6: Two 'Person' objects having different names but same address

Basic aggregation is represented using an empty diamond symbol next to the whole class as shown in figure 6.7.

Composition represents exclusive relationship between two classes. Composition is a strong type of aggregation where the lifetime of the part class depends on the existence of the owner class. If an object of aggregating class is deleted, its part class object also will get deleted. For example, when an object of class Person is deleted, the object of class Name is also deleted. Name is associated exclusively with single person as shown in figure 6.6. Composition relationships are represented using a filled diamond symbol next to the whole class as shown in figure 6.7.

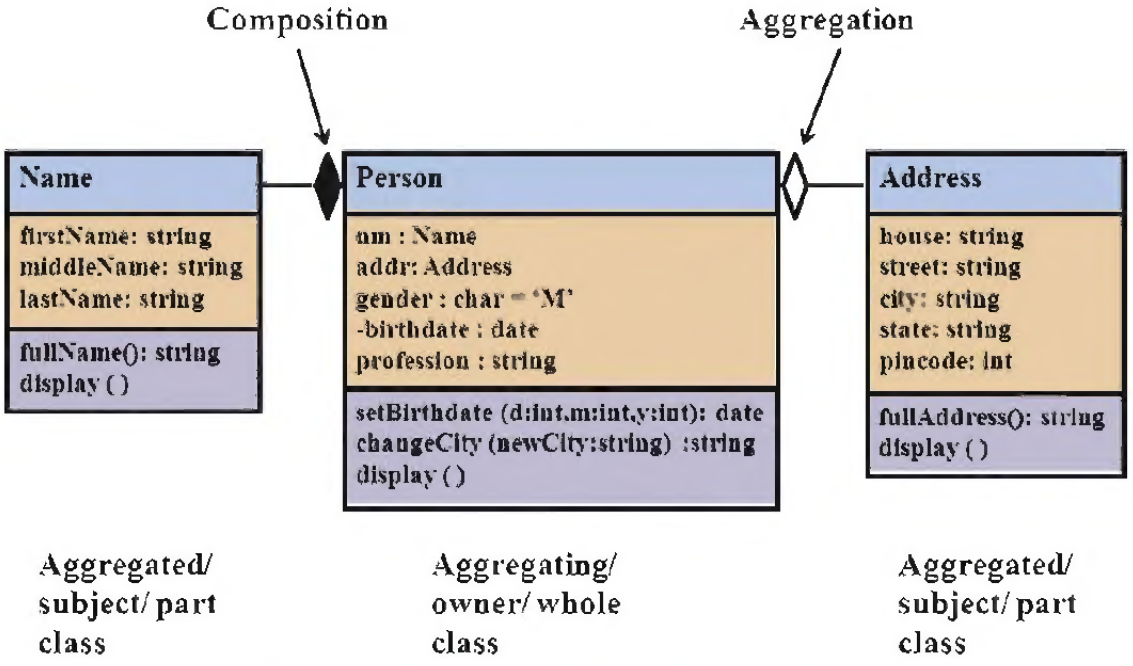

Figure 6.7 : Composition and Aggregation

In the given example, a relationship between class 'Person' and class 'Name' is composition relationship; whereas a relationship between class 'Person' and class 'Address' is aggregation relationship. Address may be shared by more than one person. So, when a person is deleted, the corresponding name object is deleted but address cannot be deleted.

Note : The class that contains objects of other class is known as owner class or whole class or aggregating class. For example, the class 'Person' shown in figure 6.7 is an aggregating class.

The class that is contained in owner class is known as subject class or part class or aggregated class. For example, the classes 'Name' and 'Address' shown in figure 6.7 are aggregated class.

Inheritance

Inheritance is generally referred to as 'is-a-kind-of' relationship between two classes. It is appropriate when one class is 'a kind of' other class. For example, teacher is a kind of person. So, all the attributes and methods of class 'Person' are applicable to class 'Teacher' also. In other words, class 'Teacher' inherits all attributes and behavior of class 'Person'. Class 'Teacher' may have additional attributes like subject and methods like taking lectures of the subject. In such scenario, class 'Teacher' can be defined using class 'Person'.

Inheritance refers to the capability of defining a new class of objects that inherits the characteristics of another existing class. In object-oriented terminology, new class is called sub class or child class or derived class; whereas the existing class is called super class or parent class or base class. The data attributes and methods of the super class are available to objects in the sub class without rewriting their declarations. This feature provides reusability where existing methods can be reused without redefining. Additionally new data and method members can be added to the sub class as an extension. It also allows the methods to be redefined in subclass as per need.

In class diagram, inheritance is represented using an arrow pointing to superclass as shown in figure 6.8. In this example, 'Person' is a superclass and 'Teacher' is a subclass.

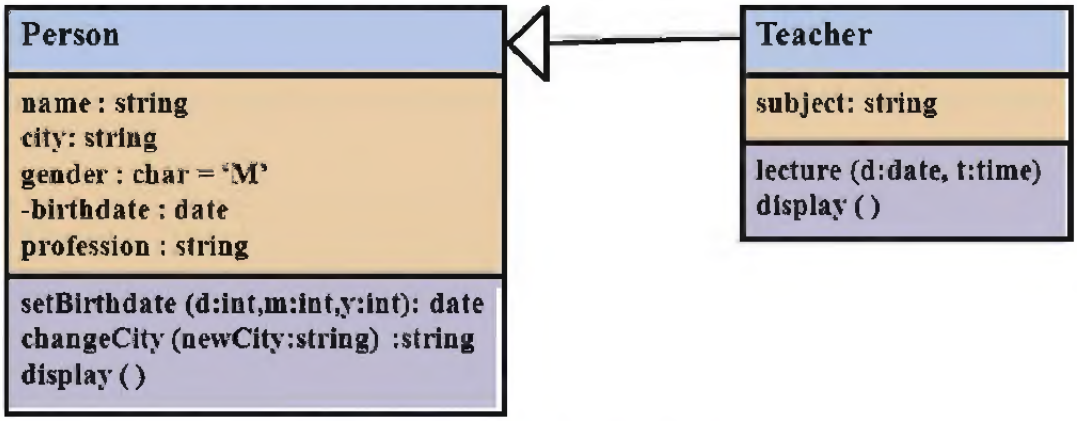

Figure 6.8 : Example of Inheritance

Generalization is another name for inheritance or "is a" relationship. It refers to a relationship between two classes where one class is a specialized version of another. Common attributes and methods are defined in superclass. Subclass is a specialized version with additional attributes and methods.

Sometimes, there may be a classical hierarchy of inheritance between classes. For example, class 'Employee' can be derived from class 'Person', then class 'Teacher' can be derived from 'Employee'. Here employee is a kind of person and teacher is a kind of employee. Such type of inheritance is known as multilevel inheritance. An example of multilevel inheritance is shown in figure 6.9.

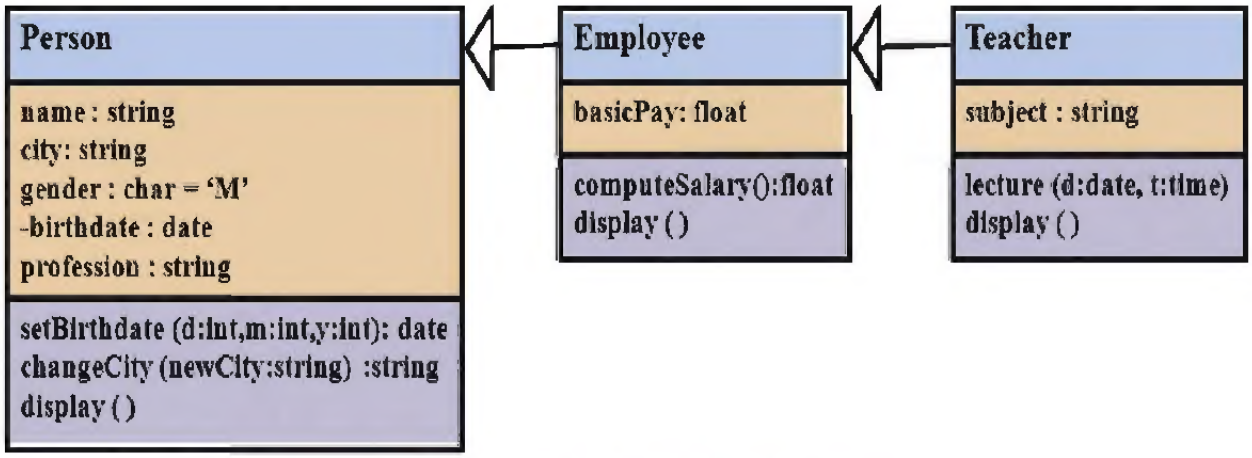

Figure 6.9 : Example of multilevel inheritance

A class can also be derived using more than parent classes. For example, a child inherits the characteristics of both mother and father; airplane is a kind of vehicle as well as flying object. When a class is derived from two or more classes, it is known as multiple inheritance.

Composition vs. Inheritance

In inheritance, class inherits from other classes in order to share, reuse or extend functionality. Here there exists 'a kind of' relationship between super class and subclass.

In composition, classes do not inherit from other classes, but are 'composed of' other classes. Class contains the attributes where some attributes are of objects of other class types.

Note that there other types of relationships and also constraints that can be represented in class diagram. The study of all the concepts is out of scope of this book.

Summary

Object-oriented as methodology is playing significant role in analysis, design and implementation of software system. In this paradigm, the focus is on objects which contain both data and functionality together. A class encapsulates attributes (data) and methods (behavior or functionality) together as a template that can be shared by all objects. Objects are then distinguished by their state; that is value of these attributes. More than one class may be associated with each other. When there is 'has-a' or 'a-part-of' relationship between two classes, the relationship is called aggregation or composition. When a class contains objects of other class, the container class is called owner class or whole class or aggregating class. The class that is contained in owner class is known as subject class or part class or aggregated class. Aggregation represents non-exclusive relationship between two classes. Composition represents exclusive relationship between two classes. When there is 'is-a' or 'a-kind-of' relationship between two classes, there is inheritance relationship. General or common features of two classes are implemented in the superclass and special features are implemented in subclass.

EXERCISE

1. List the features supported by object-oriented programming languages.
2. Distinguish between class and object.
3. Differentiate between 'Encapsulation' and 'Data Abstraction'.
4. What do you mean by polymorphism ? Name two type of overloading that may be supported to achieve polymorphism.
5. Explain the use of aggregation and composition.
6. When should one use inheritance ? Give example.
7. Explain different types of inheritance.
8. Choose the most appropriate option from those given below :
   (1) In Object-oriented methodology, the focus is on which of the following entities ?
   (a) Data (b) Functions (c) Objects (d) All of the above

(2) Which of the following best suits to Java ?

(a) A procedural programming language

(b) An Object-oriented programming language

(c) A Query language

(d) All of the above

(3) Which of the following is used to distinguish objects from each other ?

(a) Attributes (b) State (c) Behavior (d) All of the above

(4) Which of the following is used to define common features of similar objects ?

(a) Class (b) Object (c) Methods (d) All of the above

(5) Which of the following is not a visibility symbol ?

(a) ~ (b) * (c) # (d) -

(6) Which of the following is provided using encapsulation ?

(a) Data protection (b) Data sharing

(c) Separation of data and methods (d) All of these

(7) Which of the following is enabled by data abstraction ?

(a) Data protection (b) Data hiding

(c) To hide implementation details of method manipulating the data

(d) All of these

(8) With which of the following options polymorphism cannot be achieved ?

(a) Method overloading (b) Operator overloading

(c) Data hiding (d) All of these

(9) An aggregation model refers to which of the following relationships ?

(a) 'is-a' relationship (b) 'is-like' relationship

(c) 'a-part-of' relationship (d) All of these

(10) An inheritance model refers to which of the following relationships ?

(a) 'is-a' relationship (b) 'has-a' relationship

(c) 'a-part-of' relationship (d) All of these

(11) In class diagram, composition is represented using which of the following symbols ?

(a) Empty diamond symbol (b) Filled diamond symbol

(c) Empty triangle symbol (d) All of these